Algorithms and Data Structures
A13. Sorting: Lower Bound

Gabriele Röger

University of Basel

March 20, 2024

A13.1 Lower Bound on Necessary Comparison Operations
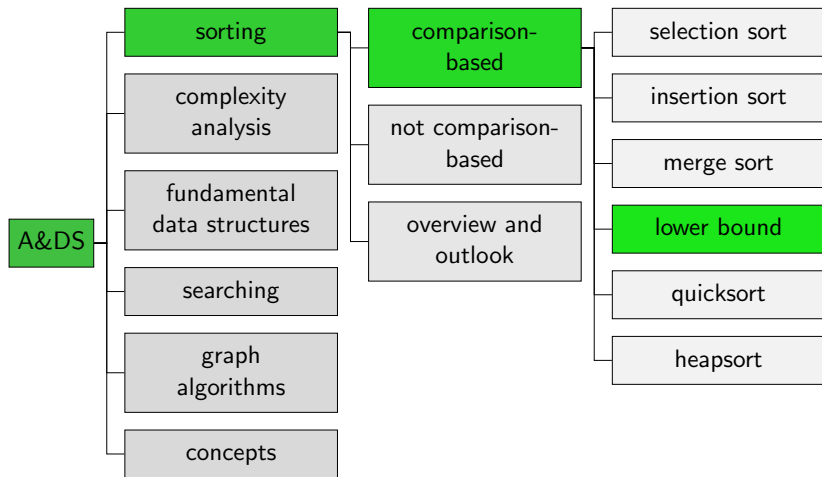
A13.2 Summary

# A13.1 Lower Bound on Necessary Comparison Operations

Content of the Course

Question

- ▶ So far, merge sort and heapsort had with $O(n \log_2 n)$ the best (worst-case) running time.
- ▶ Can we do better?
- ▶ We show: Not with comparison-based approaches!

How we Proceed

▶ Difficulty: We cannot analyze a specific algorithm but must make an argument for all possible approaches.

▶ Comparison-based approaches can only analyze the input by means of key comparisons.

▶ They must sort every input correctly.

▶ From this, we can derive a lower bound on the number of key comparisons in the worst case.

Crash Course: Binary Trees
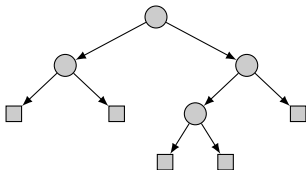
- ▶ Binary tree: each node has at most two successor nodes.
- ▶ Nodes without successors are called leaves (Image: squares).
- ▶ The node without a predecessor (at the top) is the root.
- ▶ The depth of a leaf is the
  number of edges from the root to the leaf.

<div align="center">
The maximal depth of a leaf in a binary tree
with k leaves is at least $\log_2 k$.
</div>

Exercise (Slido)
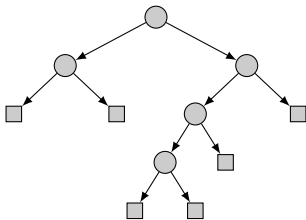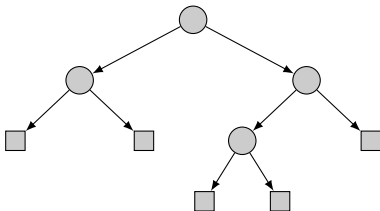

What is the maximal depth
of a leaf in this tree?

Abstract Behavior as Tree

Consider an arbitrary comparison-based sorting algorithm *A*.

▶ Its behavior only depends on the results of key comparisons.

▶ For each key comparison, there are two possibilities how the algorithm proceeds.

▶ For an input of a given size, we can depict this graphically as a decision tree.

▶ Execution of A corresponds to tracing a simple path from the root down to a leaf.

Result as Permutation

What does the algorithm have to be able to do?

▶ Assumption: all input elements distinct.

▶ Must sort all input sequences of size n correctly.

▶ We can adapt all algorithms so that they trace from which position to which position they move the elements.

▶ Then the result is not the sorted array, but the corresponding permutation.

▶ Since all possible inputs of size n must be sorted correctly, the algorithm must be able to generate all $n!$ possible permutations.

Example: Tree for Insertion Sort on Three Elements
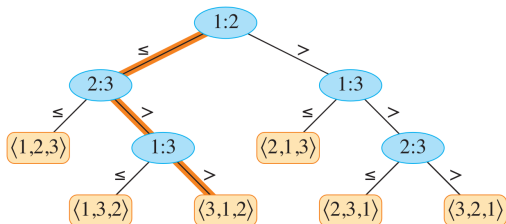
Highlighted path e.g.
for sorting sequence
$[a_1 = 6, a_2 = 8, a_3 = 5]$

Source: Cormen et al., Introduction to Algorithms

Lower Bound I

▶ Each leaf in the tree corresponds to one permutation.

▶ For input size n, the tree must thus have at least $n!$ leaves.

▶ The maximal depth of a leaf in the tree is therefore $\geq \log_2(n!)$.

▶ There is an input of size n with $\geq \log_2(n!)$ key comparisons.

Lower Bound II

Lower bound on $\log_2(n!)$

▶ It holds that $n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$

  $4! = 1 \cdot 2 \cdot \underset{\geq 2}{3} \cdot \underset{\geq 2}{4} \geq 2^2$

▶ $\log_2(n!) \geq \log_2\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right) = \frac{n}{2} \log_2\left(\frac{n}{2}\right)$

  $ = \frac{n}{2}\left(\log_2 n + \log_2 \frac{1}{2}\right) = \frac{n}{2}\left(\log_2 n - \log_2 2\right)$

  $ = \frac{n}{2}\left(\log_2 n - 1\right)$

Theorem

Every comparison-based sorting algorithm requires $\Omega(n \log n)$ *many key comparisons. As a result, also the running time is* $\Omega(n \log n)$.

Merge sort is asymptotically optimal.

A13.2 Summary

Summary

▶ Every comparison-based sorting algorithm has
at least linearithmic running time.